2. Basic Syntax

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

First Python Program

Let us execute the programs in different modes of programming.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt-

```
$ python
Python 3.3.2 (default, Dec 10 2013, 11:35:01)
[GCC 4.6.3] on Linux
Type "help", "copyright", "credits", or "license" for more information.
>>>
On Windows:
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Type the following text at the Python prompt and press Enter-

```
>>> print ("Hello, Python!")
```

If you are running the older version of Python (Python 2.x), use of parenthesis as **inprint** function is optional. This produces the following result-

```
Hello, Python!
```

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have the extension.**py**. Type the following source code in a test.py file-

```
print ("Hello, Python!")
```

We assume that you have the Python interpreter set in **PATH** variable. Now, try to run this program as follows-

On Linux

```
$ python test.py
```

This produces the following result-

```
Hello, Python!
```

On Windows

```
C:\Python34>Python test.py
```

This produces the following result-

```
Hello, Python!
```

Tokens

Token is the smallest unit of the program. Token can be defined as a punctuation mark, reserved words and each individual word in a statement.

There are following tokens in Python:

- Identifiers
- Keywords
- Literals
- Operators(See on 4. Basic Operators)

Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **age**, **Age** and **AGE** are three different identifiers in Python.

Here are naming conventions for Python identifiers-

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.

- Starting an identifier with two leading underscores indicates a strong private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Keywords (Reserved Words)

The following list shows the Python keywords. These are reserved words and you cannot use them as constants or variables or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	Not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Literals:

Literals can be defined as a data that is given in a variable or constant. Python support following literals:

1. String Literals:

String Literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a string.
E.g. "BCA", '7388313', 'Samira'

2. Numeric Literals:

Numeric Literals are immutable. Numeric Literals can belong to following four different numerical types.

Int(signed integers):

Numbers(both positive & negative) with no fractional part.
E.g. 100, -45

Long(long integers):

Integers of unlimited size followed by lowercase l or uppercase L.
E.g. 97247937L, 88493650l

Float(floating point):

Real numbers with both integer & fractional part.
E.g. -26.3, 0.314

Complex(complex):

In the form of **x+yj** where **x** forms the real part and **y** forms the imaginary part of complex number.
E.g. 2+5j, 3.14j, 4.3+2j

3)Boolean Literals:

A Boolean Literal can have any of the two values: **True** and **False**.

4)Special Literals:

Python contains one special literal **None**.
E.g.

```
>>>test=None
>>>print test
None
```

Lines and Indentation

Python does not use braces({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example-

```
if True:
    print ("True")
else:
  print ("False")
```

However, the following block generates an error-

```
if True:
    print ("Answer")
       print ("True")
else:
  print "(Answer")
print ("False")
```

```
if True:
    print ("Answer")
    print ("True")
else:
   print "(Answer")
 print ("False")
```

Thus, in Python all the continuous lines indented with the same number of spaces would form a block. The following example has various statement blocks-

Objects

Objects are the core things that python programs manipulate. Every object has a type that defines the kinds of things that programs can do with objects of that type. Types are either scalar or non-scalar.

1)Scalar Objects:

Scalar objects are indivisible. Think of them as the atoms of the language. Python have 4 types of scalar objects.

a) Int:

It is used to represent integers. Literals of type int are written in the way we typically denote integers(e.g. -3, or 313 or 2021)

b) Float:

It is used to represent real numbers. Literals of type float always include decimal point(e.g. 3.0 or 3.14 or -32.84)

c) Bool:

It is used to represent the Boolean values True or False

d) None:

It is a type with single special value i.e. None

2)Non-scalar Objects:

->They have internal structure. E.g. strings

User Input:

In Python3, input() allow for user input. This function first takes the input from the user and then evaluates the expression.

```
#!/usr/bin/python3
val=input("Enter the value:")
print(val)
```

When you execute the above program, it produces the following result-

```
Enter the value:
Samir
```

Multi-Line Statements

Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue. For example-

```
total = item_one + \
        item_two + \
        item_three
```

The statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example-

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal-

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal is the beginning of a comment. All characters after the #, up to the end of the physical line, are part of the comment and the Python interpreter ignores them.

```
#!/usr/bin/python3
```

```
# First comment
print ("Hello, Samir!") # second comment
```

This produces the following result-

```
Hello, Samir!
```

You can type a comment on the same line after a statement or expression-

```
name = "Samir" # This is again comment
```

Python does not have multiple-line commenting feature. You have to comment each line individually as follows-

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt and the statement saying “Press the enter key to exit”, and then waits for the user to take action −

```
#!/usr/bin/python3
input("\n\nPress the enter key to exit.")
```

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on a single line given that no statement starts a new code block. Here is a sample snip using the semicolon-

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```